Homework 0 - Alohomora Report.

Venkatesh Mullur
Robotics Engineering Department,
Worcester Polytechnic Institute,
Worcester,MA, USA.
vmullur@wpi.edu

I. Introduction

In this project, we learned about edge and boundary detection using classical Sobel and Canny filters, and more modern PB(Probability of Boundary) approach. In this phase, we learn about different aspects of the image which affect the boundary detection alogrithms and how can they be tuned to get better results. Apart from that, we also use machine learning techniques to perform "vector quantization" that helps to reduce the data into relevant information and discards the unnecessary data. In the second phase of this project, we learn about the deep learning models from scratch and how the different Deep Learning architectures affect the accuracy and loss per epoch.

II. Background

Having a boundary detection algorithm that focus solely on the region of interest in the image is great, but classically in image processing and computer vision, sobel, canny and prewitt were the filters used to find the edge detection. In these filters, they find the intensity discontinuations in the image that can be said as a high pass filter. It basically passes the pixel differences and supresses the low difference region. More recent algorithm called PBLite that outperforms these classical techniques using considering texture, brightness and colour differences. We are using Berkeley Segmentation Data Set 500 for deep learning. To perform the PBlite algorithm, I will be using Derivative of Gaussians(24), Leung-Malik Filters(84), and Gabor Filters(32) with a total of 140 filters.

III. Phase I

A. *Derivative of Gaussian:*

- To create a Derivative of Gaussian (DoG) filterbank, I previously convolved a sobel filter with a predefined Gaussian Filter but upon understanding the correct way, I formed a gaussian kernel using the gaussian kernel formula which looks as shown in the figure below with its equation.

- By derivative of the image, we can 2D convolve the base image by a sobel, canny or a prewitt operator.

- My gaussian function takes two standard deviations to scale the filter, by this way we can create a lot of difference of gaussian of different scales. I personally have made 24 difference of gaussian filters. Which I am showing below:

$$G(x,y) = \frac{1}{2\pi\sigma_x\sigma_y}\exp-(\frac{x^2}{2\sigma_x^2})\exp-(\frac{y^2}{2\sigma_y^2}) \quad (1)$$

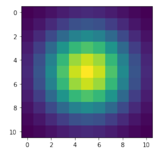


Fig. 1. Gaussian Filter Visualization

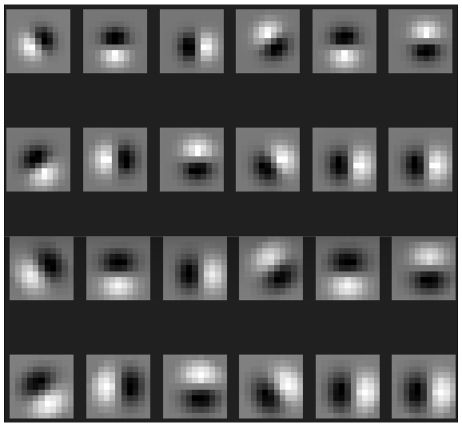


Fig. 2. Derivative of Gaussian Filter Visualization

B. *Leung-Malik Filters:*

- In Leung-Malik filters, they have 4 parts, LML, LMS, Laplacian of Gaussian and gaussians with different scales. The scales for LMS are [1,sqrt(2), 2] and LML are [sqrt(2), 2*sqrt(2), 4].

- The LM filter bank consists of total of 48 filters of LMS and 48 of LML. Out of the 48, the 36 are LM filters and the rest are either Laplacian of Gaussian or the Gaussians as shown above.

- To find the laplacian, I implemented the equation shown below and got the result as shown after the equation. By changing the scales as given above, we can change the standard deviation of the Laplacians as well as the gaussians.

$$L(x,y) = -\frac{1}{\pi\sigma^4}[1 - \frac{x^2+y^2}{2\sigma^2}]\exp -(\frac{x^2+y^2}{2\sigma^2}) \tag{2}$$

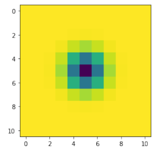


Fig. 3. Laplacian of Gaussian Filter Visualization

- The LML and LMS filterbank consists of first order derivative of the gaussian and the second order derivative. As discused above, the second order derivative of the gaussian filter can be obtained by convolving the sobel operator twice on the main image which is the gaussian having different scales.

- By, changing scales, essentially changing the standard deviation, rotation and convolving once or twice depending on the order of derivative we want, I created a total of 84 filters which are shown below.

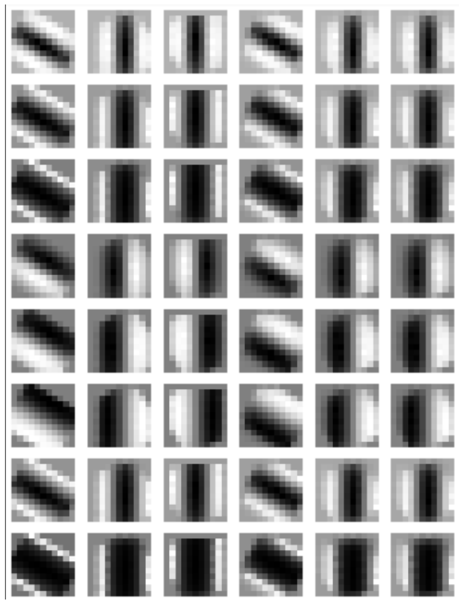


Fig. 4. Leung Malik Filters Visualization

C. Gabor Filter:

- Gabor Filters are present in the human eye and are very important for feature extraction, they are created by modulating a sin wave with variable frequency and a gaussian filter with variable standard deviation.

- These filters are created to detect the effect of a Gabor filter and the texture differences in the image.

- The shape of the gabor filter is dependant on 5 variables sigma, theta, lambda, psi and gamma. Sigma is the standard deviation, theta is the angle of rotation, lambda is the wavelength, psi is the offset, and gamma is the measure of ellipticity. The gabor filters of Lambda = 5, psi = 1.5, gamma = 0.75, sigma = 6, theta = pi/2 is shown below.

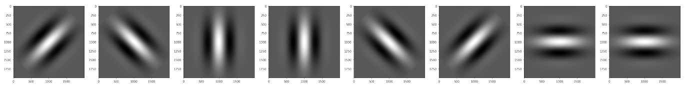


Fig. 5. Gabor Filters of Lambda = 5, psi = 1.5, gamma = 0.75, sigma = 6, theta = pi/2 Visualization

D. Texton, Brightness, Color Maps:

- To find the Texton, Brightness and Color maps, we convolve all the filters from the previous bank with the image. We use grayscale of the image for finding texton and brightness maps while a colored image to find the color map.

- In my case, I had created 140 filters in the filter bank and my image size is (321, 481). After convolving the image with 140 filters, the result I got was of the size (321,481,140) . To reduce the dimensionality from 140 to 64, we use Kmeans Clustering having 64 clusters.

- Thus, each pixel in the filtered image had 140 features and after Kmeans clustering, it was reduced to 64. That means each pixel in the image could have 140 values but then it was reduced to 64 using clustering. To use this in the ”sklearn.cluster.KMeans”, you need to have the data in a 2D format. To perform this, I reshaped the data using numpy in (321*481,140) form. Then this reshaped data is given to the KMeans. The predicted output of the clustering when our reshaped data is the input, is the texton map. This texton map needs a reshape back to its original form (321,481) to form the actual texton map.
- The texton map in my project looks like the image below.

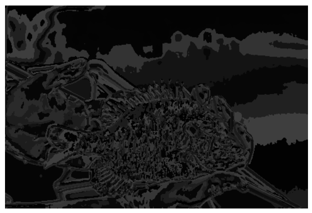


Fig. 6. Texton Map Visualization

- The grayscale of the original image is directly given to the clustering algorithm to get a brightness map. In this map, it finds out the brightness discontinuities and highlights them and is shown in the image below.

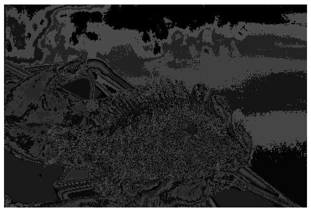


Fig. 7. Brightness Map Visualization

- To create the colormap, that points out the color differences in the image, I did the same process I did to create a brightness map, but since it is a color map, I performed the same clustering on 3 primary colors namely R,G and B.
- So essentially, I got 3 different colormaps. After recombining them to form the colormap in RGB format, it looks as the image shown below.

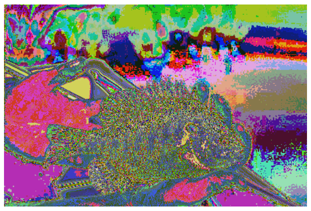


Fig. 8. Color Map Visualization

E. Half Disks:

- Half disks are just some filters having two contrast sides of the same image that essentially compare the distributions on both sides of the filters. They are created in different scales and orientations are shown below:

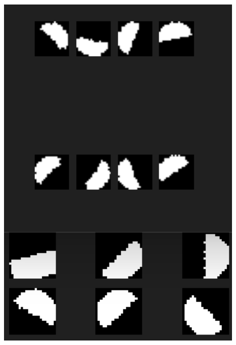


Fig. 9. Half Disks Visualization

- Texture gradients, brightness gradients and color gradients are formed by convolving all the half disks filters with their maps respectively.

- Ultimately since we have deliberately made the filter bank of the half disk in such a way that they are in the form of pairs. That means, every pair contains a half disk and a flipped half disk.
- The texture gradient is shown in the fig 10 given below.

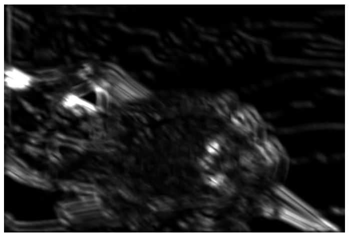


Fig. 10. Texton Gradient Visualization

- This helps to find the distributions on the image when convolved woth half disk filter bank.
- If there is a high discontinuity in the image, then the gradient will be high but if the discontinuity is low, then the gradient will also be low.
- The brightness gradient is shown below in the figure 11.

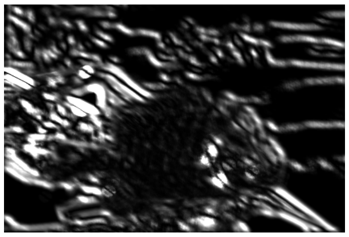


Fig. 11. Brightness Gradient Visualization

- Because of the half-disk filter bank, they span multiple scales and orientations, it will result in a series of local gradient measurements encoding how quickly the texture or brightness distributions are changing at different scales and angles.
- The color gradient is shown in the fig 12 below.

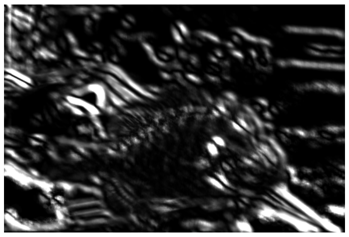
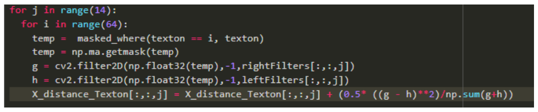


Fig. 12. Color Gradient Visualization

F. Chi Square Distance:

- It calculates the distance between 2 histograms. Here the two histograms are the result of convolution of every gradient map with the left half and the right half of the half disk.

- The convolution is performed using cv2.filter2D() function in OpenCV, and the image below shows how I calculated the difference of histogram. Here g is the left histogram and h is the right histogram.



```
for j in range(14):
  for i in range(64):
    temp =  masked_where(texton == i, texton)
    temp = np.ma.getmask(temp)
    g = cv2.filter2D(np.float32(temp),-1,rightFilters[:,:,j])
    h = cv2.filter2D(np.float32(temp),-1,leftFilters[:,:,j])
    X_distance_Texton[:,:,j] = X_distance_Texton[:,:,j] + (0.5* ((g - h)**2)/np.sum(g+h))
```

Fig. 13. Snippet to calculate the Chi square distance.

$$X^2(g,h) = \frac{1}{2}[\sum_{i=1} \frac{(g_i - h_i)^2}{(g_i + h_i)}] \tag{3}$$

- The above snippet shows how I calculated the chi squared distance and the formula is given above.

G. PBLite Final:

- PBCanny and PBsobel boundary detection was already provided by the professor. w1 and w2 were the weights given to them repectively. I tried with w1, w2 = 0.5 and it gave pretty good results.

- Below equation shows the formula to be referred to get the final PBlite output which clearly is better than every individual output.

$$PBLite = \frac{T_g + B_g + C_g}{3} \odot (w1 * cannyPB + w2 * sobelPB) \tag{4}$$

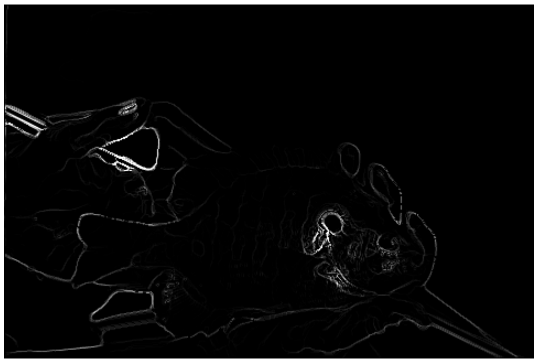


Fig. 14. PBLite Final Visualization.

- The main advantage of using this process was to find the edges or boundaries in the region of interest inside the image. With classical filters, we might get the edges of the things which are out of focus in the background

IV. Phase II

In this phase, I try to implement a custom network made by me, resnet, resnext and dense net. Due to the limited computational capacity and since I am new to deep learning, I might have missed some things.

A. Custom Network:

- I tried to implement a network similar to VGG16, but due to small size of the images in the CIFAR dataset, I had to limit the architecture.

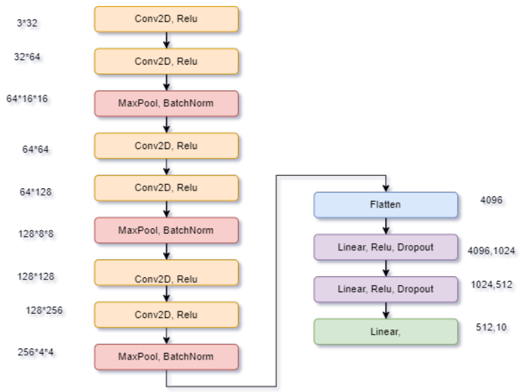


Fig. 15. CustomNet Visualization.

- The parameters are:
- Number of parameters = 5,446,666
- Optimizer = SGD
- Learning Rate = 0.0001
- Batch Size =
- Epochs = 10
- Training Accuracy = 82.6
- Testing Accuracy = 36.8

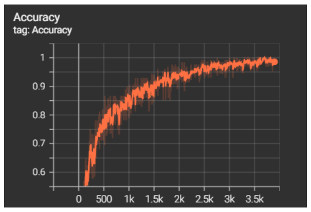


Fig. 16. Model Summary.

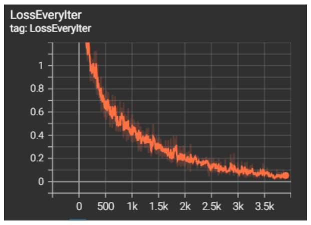
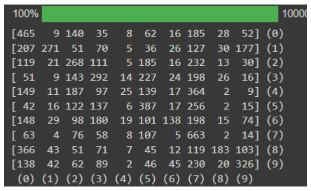
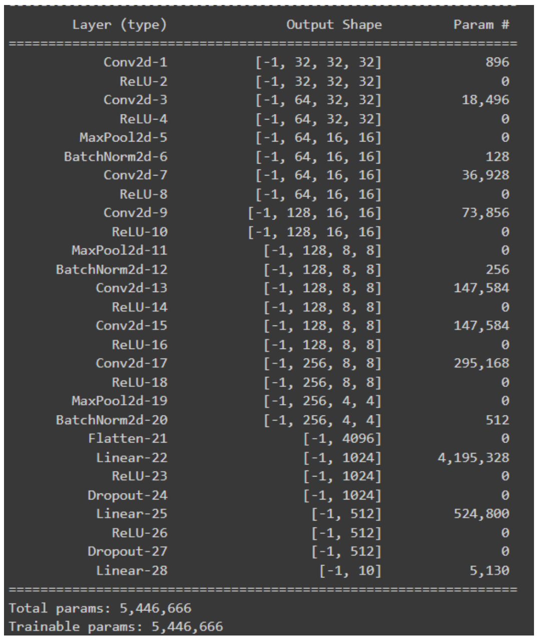


Fig. 17. Model Summary.



```
100%                                          10000
[465   9 140  35   8  62  16 185  28  52] (0)
[207 271  51  70   5  36  26 127  30 177] (1)
[119  21 268 111   5 185  16 232  13  30] (2)
[ 51   9 143 292  14 227  24 198  26  16] (3)
[149  11 187  97  25 139  17 364   2   9] (4)
[ 42  16 122 137   6 387  17 256   2  15] (5)
[148  29  98 180  19 101 138 198  15  74] (6)
[ 63   4  76  58   8 107   5 663   2  14] (7)
[366  43  51  71   7  45  12 119 183 103] (8)
[138  42  62  89   2  46  45 230  20 326] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
```

Fig. 18. Model Summary.



```
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 32, 32]             896
              ReLU-2           [-1, 32, 32, 32]               0
            Conv2d-3           [-1, 64, 32, 32]          18,496
              ReLU-4           [-1, 64, 32, 32]               0
         MaxPool2d-5           [-1, 64, 16, 16]               0
       BatchNorm2d-6           [-1, 64, 16, 16]             128
            Conv2d-7           [-1, 64, 16, 16]          36,928
              ReLU-8           [-1, 64, 16, 16]               0
            Conv2d-9          [-1, 128, 16, 16]          73,856
             ReLU-10          [-1, 128, 16, 16]               0
        MaxPool2d-11            [-1, 128, 8, 8]               0
      BatchNorm2d-12            [-1, 128, 8, 8]             256
           Conv2d-13            [-1, 128, 8, 8]         147,584
             ReLU-14            [-1, 128, 8, 8]               0
           Conv2d-15            [-1, 128, 8, 8]         147,584
             ReLU-16            [-1, 128, 8, 8]               0
           Conv2d-17            [-1, 256, 8, 8]         295,168
             ReLU-18            [-1, 256, 8, 8]               0
        MaxPool2d-19            [-1, 256, 4, 4]               0
      BatchNorm2d-20            [-1, 256, 4, 4]             512
          Flatten-21                 [-1, 4096]               0
           Linear-22                 [-1, 1024]       4,195,328
             ReLU-23                 [-1, 1024]               0
          Dropout-24                 [-1, 1024]               0
           Linear-25                  [-1, 512]         524,800
             ReLU-26                  [-1, 512]               0
          Dropout-27                  [-1, 512]               0
           Linear-28                   [-1, 10]           5,130
================================================================
Total params: 5,446,666
Trainable params: 5,446,666
```

Fig. 19. Model Summary.